Algemeen

voorwaardelijke expressies

Als je de waarde die je aan een variabele wil toekennen laat afhangen van een bepaalde voorwaarde, dan kan je hiervoor een voorwaardelijke opdracht gebruiken. Stel bijvoorbeeld dat je werkt met een variabele `x` die verwijst naar de lengte van een persoon (in centimeter) en dat je aan de variabele `lengte` de waarde `'groot'` wil toekennen als $x > 185$ en anders de waarde `'klein'`. Met een voorwaardelijke opdracht kan je dit op de volgende manier realiseren.

```
>>> x = 100
>>> if x > 185:
...     lengte = 'groot'
... else:
...     lengte = 'klein'
...
>>> lengte
'klein'

>>> x = 190
>>> if x > 185:
...     lengte = 'groot'
... else:
...     lengte = 'klein'
...
>>> lengte
'groot'
```

Je kan hetzelfde realiseren door gebruik te maken van een *voorwaardelijke expressie* (ook een *if-else-expressie* genoemd). In tegenstelling tot een voorwaardelijke opdracht is dit dus een expressie (die een waarde oplevert) en geen statement. Hierdoor kan je een voorwaardelijke expressie zonder problemen gebruiken als rechterlid van een toekenningsopdracht. De bovenstaande interactieve sessie kan dus korter geschreven worden als

```
>>> x = 100
>>> lengte = 'groot' if x > 185 else 'klein'
>>> lengte
'klein'

>>> x = 190
>>> lengte = 'groot' if x > 185 else 'klein'
>>> lengte
'groot'
```

Testen of waarde van variabele behoort tot een vast aantal opties

De volgende voorwaardelijke opdrachten bevatten een logische fout bij het controleren of een gegeven werkdag in het weekend valt of een werkdag is. Eigenlijk is het zo dat met deze formulering de voorwaarde altijd waar zal zijn, waardoor de suggestie gewekt wordt dat het altijd weekend is.

```
>>> weekdag = 'zondag'
>>> 'weekend' if (weekdag == 'zaterdag' or 'zondag') else 'werkdag'
'weekend'
```

```
>>> weekday = 'maandag'
>>> 'weekend' if (weekday == 'zaterdag' or 'zondag') else 'werkdag'
'weekend'
```

De reden waarom het hier fout loopt is dat de voorwaarde is samengesteld uit twee deelvoorwaarden, namelijk `weekdag == 'zaterdag'` en `'zondag'`. De tweede deelvoorwaarde is altijd waar, omdat elke string waar is, behalve de lege string die niet waar is. De correcte formulering van de samengestelde voorwaarde is

```
>>> weekdag = 'zondag'
>>> 'weekend' if (weekdag == 'zaterdag' or weekdag == 'zondag') else 'werkdag'
'weekend'

>>> weekday = 'maandag'
>>> 'weekend' if (weekdag == 'zaterdag' or weekdag == 'zondag') else 'werkdag'
'werkdag'
```

Let hierbij op de herhaling van de variabelenaam in de voorwaarde.

Vermijden om meerdere `print` statements te gebruiken

Het is altijd een goed idee om ervoor te zorgen dat je programma maar één print statement bevat. Zo is het makkelijker om een overzicht te behouden van wat er allemaal uitgeschreven kan worden.

Stel bijvoorbeeld dat je volgende code hebt

```
>>> x = 3
>>> if x < 5:
...     print('kleiner dan 5')
... else:
...     print('groter dan 5')
...
'kleiner dan 5'
```

Dan herschrijf je dit beter naar

```
>>> x = 3
>>> if x < 5:
...     resultaat = 'kleiner'
... else:
...     resultaat = 'groter'
...
>>> print(f'{resultaat} dan 5')
'kleiner dan 5'
```

Landingsbaan

Voorloopnullen toevoegen met f-strings

Als je bij het omzetten van een getal naar een string wil zorgen dat de string een vast aantal karakters heeft (waarbij er eventueel nullen worden toegevoegd aan het begin van de string), dan kun je gebruikmaken van de

een *format specifier*. *Format specifiers* worden altijd tussen het paar accolades geplaatst dat gebruikt wordt als plaatshouder in de template string. Een *format specifier* begint altijd met een dubbelpunt (`:`).

Om een getal uit te schrijven dat een vast aantal posities inneemt, gebruik je de *format specifier* `:0nd`. De `0` staat er omdat we vooraan nullen willen toevoegen (je kan ook andere karakters toevoegen), de letter `d` staat voor een digit (getal) en `n` voor de lengte die de string moet innemen. Als het getal dat je wil omzetten naar een string langer is dan de gewenste lengte, dan wordt het volledige getal overgenomen. In dat geval zal de string dus langer zijn dan het gewenst aantal karakters.

```
>>> f'{2}'
'2'
>>> f'{2:02d}'
'02'
>>> f'{34:02d]'
'34'
>>> f'{567:02d}'
'567'
>>> f'{89:06d}'
'000089'
```

Er zijn nog andere manieren om voorloopnullen te genereren. Zo kan je ook gebruik maken van de string-methode `zfill`. Je kan ook het verschil tussen de huidige lengte en de gewenste lengte berekenen en dan weet je hoeveel nullen je moet toevoegen. Of je kan ook werken met een `while` lus die nullen blijft toevoegen totdat de string de gewenste lengte heeft.

```
>>> gewenste_lengte = 3
>>> getal = str(2)
>>> getal.zfill(gewenste_lengte)
'002'
>>> aantal_nullen = gewenste_lengte - len(getal)
>>> aantal_nullen
2
>>> '0' * aantal_nullen + getal
'002'
>>> while len(getal) < gewenste_lengte:
...     getal = '0' + getal
...
>>> getal
'002'
```

Vingervlug

Controleren of een getal tussen twee grenzen ligt

In Python kan je controleren of een waarde (van een variabele) binnen een interval gelegen is door gebruik te maken van een samengestelde Booleaanse expressie. Als je bijvoorbeeld wil controleren of het getal `x` in het interval $]a, b[$ ligt, dan kan je daarvoor de volgende Booleaanse expressie gebruiken

```
>>> a < x and x < b
```

In de wiskunde zou je deze voorwaarde echter schrijven als $a < x < b$ en daarom kan je in Python evengoed de volgende verkorte expressie gebruiken.

```
>>> a < x < b
```

Hieraan is duidelijk te zien dat Guido Van Rossum, de uitvinder van Python, een opleiding in de wiskunde genoten heeft. Ook de voorwaarde $a \leq x \leq b$ waarbij de grenzen behoren tot het interval kan je op dezelfde manier in Python schrijven als

```
>>> a <= x <= b
```

In Python is het een fout om de waarde van een variabele op te vragen als die variabele nog niet gedefinieer is (er is nog geen waarde toegekend aan die variabele). Dit doet zich bijvoorbeeld voor in het onderstaande codefragment

```
>>> x = 4
>>> if x < 3:
...     var = 'ok'
...
>>> print(var)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'var' is not defined
```

waar bij het `print` statement de foutmelding gemaakt wordt dat de variabele `var` niet kan teruggevonden worden in de namespace. Als je de voorgaande code goed bekijkt, dan zie je dat er enkel een waarde wordt toegekend aan de variabele `var` indien de waarde van `x` groter is dan 3 (en dat is hier duidelijk niet het geval).

Kaartmakkers

Toekenningsopdracht vs. gelijkheidstest

In Python maak je bij een toekenningsopdracht gebruik van één enkel gelijkteken en bij een gelijkheidstest (nagaan of twee objecten dezelfde waarde hebben) van twee opeenvolgende gelijktekens. Om te testen of de variabele `x` gelijk is aan twee schrijf je dus

```
>>> if x == 2:    # correct
...     pass
```

en niet

```
>>> if x = 2:     # verkeerd
  File "<myscript.py>", line 1
    if x = 2:
         ^
SyntaxError: invalid syntax
```